OhjyA & Its Algorithm
(Version 0.5 beta 1)

Ghasan Sabri Ahmed Al-Sakkaf
(I'll learn from Al-Melh bazaar and exceed you!)

To my precious family
and MSOMS community

And to all who believes in knowledge

Contents

Preface

In the Name of Allah.
Allah says: "*Allah will exalt those who believe among you, and those who have knowledge, to high ranks.*" (Al-Mujadilah, Verse 11, The Holy Quran)

From the preceding verse and the importance of knowledge, I've decided to license OhjyA (Arabic: Puzzle) program under GNU General Public License version 3, so everyone gets the right to participate and learn.

OhjyA is an ASCII-text file encryption program. You can understand ASCII as just English letters and common marks.

The idea of OhjyA popped in my mind when programming lecturer was explaining about sorting algorithms. But I, at the time, wasn't able to write OhjyA because of lack of tools and techniques I'd learned in C++. That was five month far from now. Thanks Allah, after learning some of the fundamentals of C++, I'm now able to apply the idea which was taking my attention.

Allah willing, I'll explain in this e-booklet OhjyA and its algorithm, and how to choose better passwords for better encryption, and various points regarding OhjyA and its algorithm.

Ghasan Al-Sakkaf
15 November 2010

OhjyA: The Kernel

Sometimes you come to password protected archives, and when you enter wrong password, a message appears telling you that. “Huh! How dose it know that I entered a wrong one?!” I wondered. That’s how the question was increasing in my mind. Then, I began reading from internet about algorithms of password recovery tools. The kernel idea was that the program generates numerous passwords and checks them one by one. If the password fails, the program writes it to a file to avoid generating it again, and then checks the next.

“What do you mean by “password fails”?” you may wonder. Indeed, I don’t know. But I’m sure that the program checks it so fast, even though the archive could be in gigabytes! “Then, it checks just from small part of the archive, but not all.” that’s how I thought about it. In simple, the program doesn’t go through tough process to check whether the password is correct or not, and just checks small part of encrypted archive. That’s how it started.

OhjyA is built on ‘Lakhata’ (Arabic: No Error) idea. “OhjyA should accept any password, and, at the same time, should be tough enough so unauthorized persons can’t view the message.” that’s Lakhata in a nutshell. “How dose it accept any password?” you may wonder. Lakhata idea is to accept any password and then use it to decrypt the message. So, it will decrypt the message in unreadable form. I’ll give an example to make it clear.

Example:
If Massooud phoned his father and gave him a number (Password), and told his father to sum it with the encrypted total money he sent earlier time by email to get the actual amount of money Massooud collected. If you know that the encrypted total money is 100,000 YER, how much money Massooud collected?

Answer:

Massooud’s father will easily sum the number with encrypted total money, and get the actual amount.
But you, how would you find it out?

If we supposed that the number is -100,000, then Massooud collected nothing;
-100,000 + 100,000 = 0 YER.
But you, how would you find it out?

We see clearly from the preceding example how the number (Password) has patterned the actual amount, and each one is defining the other in one encrypted data ‘100,000 YER’. So, unauthorized persons can still generate passwords, but how would they find if it’s correct or not? This is Lakhata!

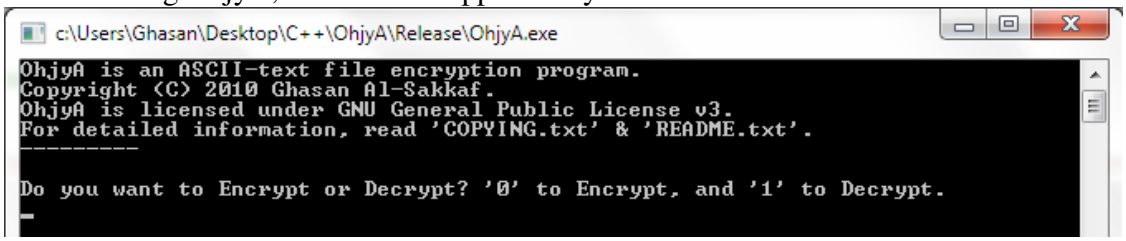
OhjyA: The Program

Before I go further and explain OhjyA algorithm, I'll explain how to use the program.

First, OhjyA is a console application, that is, it doesn't have windows nor eye-candy buttons. But don't panic, I program it just to be easy! With little thinking, you'll figure it.

Encrypt or Decrypt?

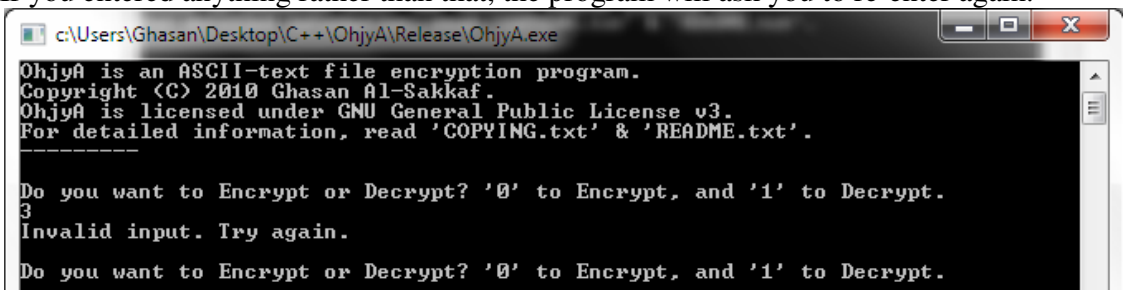
When running OhjyA, this window appears to you.



The first four lines describe some information about OhjyA.

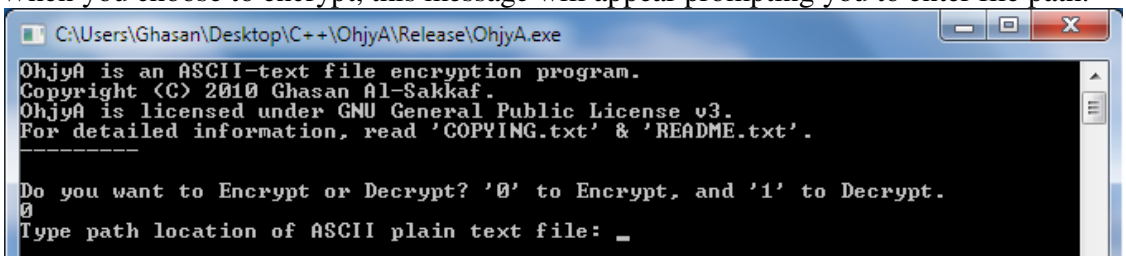
The fifth line asks you if you want to encrypt or decrypt. Press '0' followed by ENTER to encrypt, or Press '1' followed by ENTER to decrypt.

If you entered anything rather than that, the program will ask you to re-enter again.



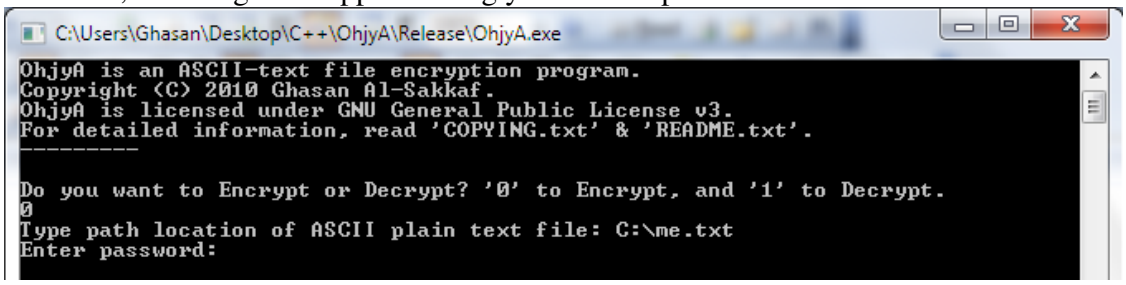
Encryption

When you choose to encrypt, this message will appear prompting you to enter file path.



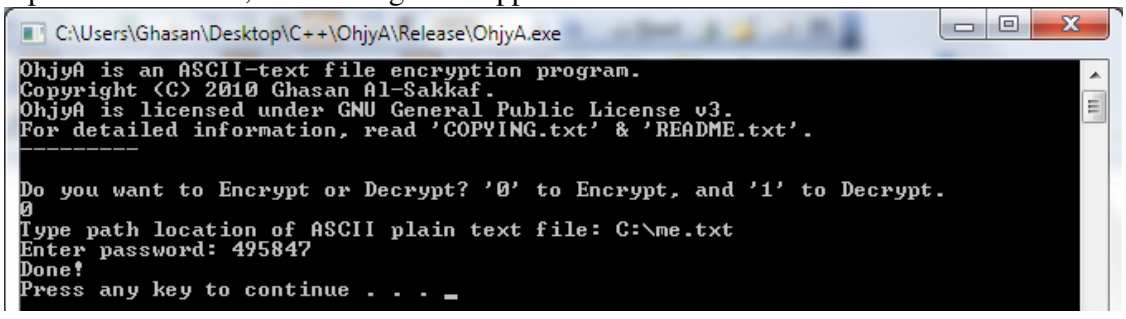
Remember that file path should be all in ASCII characters.
If you entered wrong file path, or the file doesn't exist. A message will appear prompting you re-enter file path again.
// Always include ASCII-text files with minimum of 10 characters, or OhjyA will reject it and tell you to re-enter another one. (Spaces and new lines are calculated)

After that, a message will appear telling you to enter password.



Password length should be of minimum 6 characters, and should be just numbers from 0-9.
// Password length should be of *minimum* 6 characters, but not limited to! (Refer to 'Useful Tricks' page)

If password is valid, this message will appears.

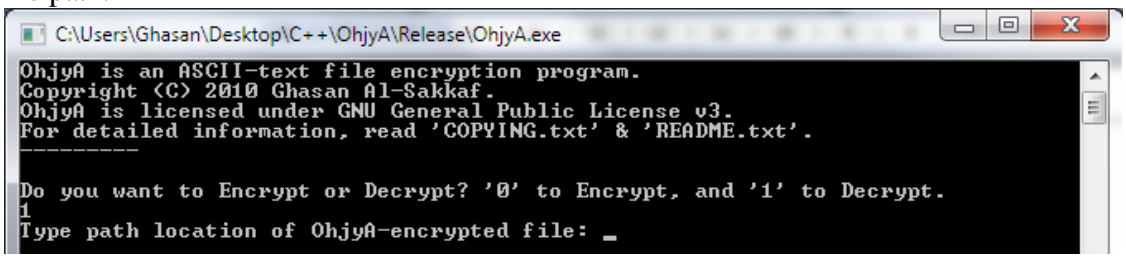


Notice that OhjyA will create a new file in the same path, with the same name, but will add '.ohj' extension to it. If encrypted file exists, OhjyA will destroy all information inside it, and re-write again over it.

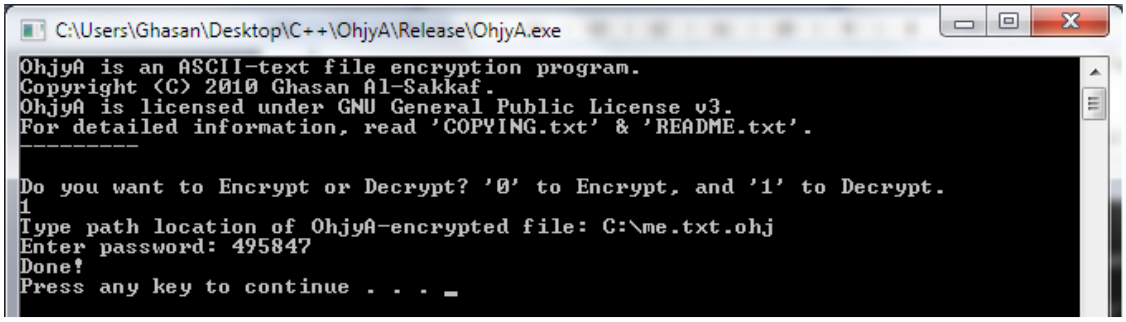
Finally, just press any key, and OhjyA will exit.

Decryption

When you choose to decrypt, this message will appear prompting you to enter encrypted file path.



Enter encrypted file path considering conditions explained in 'Encryption' part.
// It's not important that encrypted file has '.ohj' extension, but having so avoid some errors.
If the encrypted file has been modified, OhjyA may reject it.



If OhjyA accepted the encrypted file, you'll be prompted to enter password. (Same conditions in 'Encryption' part applied) No matter if the password is right or wrong, as long as it meets the conditions, OhjyA will use the password to decrypt the file, and the output will depend on the password entered!

After that, OhjyA will create a file in the same path, with same name, but will delete '.ohj' extension. If file exists, OhjyA will destroy all information inside it, and re-write again over it.

Finally, just press any key, and OhjyA will exit.

OhjyA: The Algorithm

OhjyA carries a unique algorithm inside it. Sometimes I use OhjyA to refer to algorithm, because algorithm itself is OhjyA!

Redefining

First, OhjyA redefines ASCII characters as shown below:

- A-Z are defined progressively from 11 – 36. So, A=11 till Z=36.
- a-z are defined progressively from 37 – 62. So, a=37 till z=62.
- 0-9 are defined progressively from 63 – 72. So, 0=63 till 9=72.
- Other characters is defined as the following:
    - [\n] (ENTER/New Line) = 73;
    - [ ] (Space) = 74;
    - [?] (Question) = 75;
    - [!] (Exclamation) = 76;
    - [&] (Ampersand) = 77;
    - [:] (Colon) = 78;
    - [.] (Dot/Full stop) = 79;
    - [,] (Comma) = 80;
    - [;] (Semicolon) = 81;
    - [(] (Left round bracket) = 82;
    - [)] (Right round bracket) = 83;
    - [/] (Forward slash/Solidus) = 84;
    - [+] (Add) = 85;
    - [-] (Subtract) = 86;
    - [*] (Multiply) = 87;
    - [‘] (Single quote) = 88;
    - [\] (Backslash) = 89;
    - [Others] (Any character out of listed above) = 90

// Notice that any character out of listed ones is defined with same numeric value 90. You’ll know later why I stopped at 90.

Addition

Before forwarding, I think it's a good idea to include an example to play with!
Let's suppose that user entered a file containing this string "ABCDEFGHIJ", so the redefined string would be "11121314151617181920". Also, let's suppose that password string is "123456".

OhjyA loops over redefined string one by one, and at the same time, loops over password string. When looping over both redefined string and password string OhjyA sums them together and produce a new value instead of perspective element in redefined string.

Let's see how it works below:

1. Takes 11 + 1 = 12
2. Takes 12 + 2 = 14
3. Takes 13 + 3 = 16
4. Takes 14 + 4 = 18
5. Takes 15 + 5 = 20
6. Takes 16 + 6 = 22
7. Takes 17 + 1 = 18 (OhjyA turns over password string again when it ends)
8. Takes 18 + 2 = 20
9. Takes 19 + 3 = 22
10. Takes 20 + 4 = 24

Then, redefined string becomes "12141618202218202224".

Storm

'Storm' technique is the corner stone on which OhjyA relies. It changes the positions of redefined string in unpredictable way!

As we have seen till know, OhjyA is treating with redefined string as 2-decimal-position, that is, 20, for example, is treated as whole. Strom, on the other hand, separates them into 1-decimal-position numbers, that is, 20 is treated as 2 and 0, and treats them separately from each other.

Strom plays around with 1-decimal-position redefined string and moves elements to other positions depending on the value of password string.

Till now, redefined string is "12141618202218202224", let's go!

1. Moves 1 by 1 step: 2114161820221820224
2. Moves 1 by 2 steps: 2411161820221820224
3. Moves 1 by 3 steps: 2461111820221820224
4. Moves 1 by 4 steps: 2468111120221820224
5. Moves 1 by 5 steps: 2468011121221820224
6. Moves 1 by 6 steps: 2468021121211820224
7. Moves 1 by 1 step: 2468021121211820224
8. Moves 1 by 2 steps: 2468021121211820224
9. Moves 2 by 3 steps: 2468021111221820224
10. Moves 1 by 4 steps: 2468021118221120224
11. Moves 2 by 5 steps: 2468021118021122224
12. Moves 2 by 6 steps: 2468021118021122224
13. Moves 1 by 1 step: 2468021118021122224
14. Moves 1 by 2 steps: 2468021118021221224
15. Moves 2 by 3 steps: 2468021118021221224
16. Moves 1 by 4 steps: 2468021118021224221
17. Doesn't move 2 by 5 steps: 2468021118021224221
18. Doesn't move 2 by 6 steps: 2468021118021224221
19. Moves 1 by 1 step: 2468021118021224221

Notice when position is out of string limit, OhjyA ignores it and continue to the next element of both redefined and password string.

Redefined string is now "2468021118021224221".

Numbers Mask

Numbers Mask is a technique of changing numbers by other ones.

First, OhjyA creates an integer array of 10 elements carries 0-9 numbers, that is, every position carries the same value of its position. That's to say, position 0 = 0, position 1 = 1, and so on. This is shown in the table below.

0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9

OhjyA then performs Strom on integer array:

1. Moves 0 by 1 step: 1023456789
2. Moves 0 by 2 steps: 1320456789
3. Moves 2 by 3 steps: 1350426789
4. Moves 0 by 4 steps: 1357426089
5. Moves 4 by 5 steps: 1357926084
6. Moves 2 by 6 steps: 1357926084
7. Moves 6 by 1 step: 1357920684
8. Moves 6 by 2 steps: 1357920486
9. Doesn't move 8 by 3 steps: 1357920486

Now, array elements were shifted as the following:

1	3	5	7	9	2	0	4	8	6
0	1	2	3	4	5	6	7	8	9

Every number inside redefined string will be masked as the previous table indicates. So, redefined string will be changed form "24680211180212242212" to "59081533381535595535".

Here is where OhjyA ends, and write redefined string inside the encrypted file.

Decryption is an inversed process of encryption algorithm. But the difference will happen in undefined characters, which will be treated as on character, and will be decrypted to '@' character.

I stopped the definition at 90, because of Strom technique. If I exceeded 90, let's say 91, and password element 9 come across it, the sum of both would be 100. 100 is 3-decimal-postion number, whereas Storm, at least for the current stage, is able just to treat 2-decimal-position number.

I hope OhjyA will not stop here, but will be developed further. That's why I licensed as free software; to give people the right to participate and learn!

OhjyA: Pros & Cons

Pros:

- Speed: OhjyA is so fast especially when comes to large files.
- Lakhata: is a great plus to OhjyA, by which, unauthorized persons will have to go through whole encrypted file when checking passwords, furthermore, they have to use text detection tool to make sure if output is readable text. (This even could be avoided. Refer to 'Useful Tricks' page)
- OhjyA is written in Native C++, which means it can be build on other platforms such as: Unix, Linux, Mac OS, Solaris, and…
- OhjyA is an open-source program under GNU/GPLv3. (Read 'COPYING.txt' file included with OhjyA bundle for detailed information)
- OhjyA kernel is unique, yet strong and extendible! (Indeed, I have a hellish idea popping in my mind right now)
- OhjyA can be used with other projects regarding chat clients, proxy websites, and so on.
- If you find it weak, develop it yourself, and don't wait anyone to do so. Because, in simple, you've incredible ideas inside!

Cons:

- Encryption of small files is weaker than large ones. The larger the file, the stronger the encryption.
- If password is a sequential numbers such as: 123456 and 456789 especially with small files, then unauthorized decryption possibilities increases. (It's okay for: 124567 and 456879)
- Self-Looping passwords such as: 73637363 and 6848968489 are treated if they were: 7363 and 68489, because OhjyA itself loops over the password when it ends.

OhjyA: Useful Tricks

Some OhjyA cons could be overcome as explained below:

- If the file you're meaning to encrypt is large, it's good idea to use long password. But also, make sure it's not weak one as explained earlier.
- To avoid text detection tools, you are encouraged to use SMS language. That is, instead of writing: Wait for me at the foyer, you can write: w8 4 me @ the foyr.
- It's also possible to encrypt the encrypted file, but when doing so, it's good to change password used for first encryption, even by just one number. And then, you'll have to decrypt twice to get the actual text.

OhjyA: Beyond 0.5 beta 1

I'll not stop, Allah willing, developing OhjyA. This is just the beginning for me, and I hope that persons from all over the world will develop OhjyA to be just unique as it's meant to be!

I may be like a kid when saying I'll make OhjyA the best, but that is really what I'm attending to do. I also hope that OhjyA would be applied in non-text files!

At the mean time till next New Year, I'll be focusing on enhancing OhjyA algorithm and deploying some ideas on it.

Words of Appreciation

OhjyA is not just a result of my hard work, but also, it's because people inspires me along the way.

I'd like to express my compliments to:

- My precious family.
- Mohammed Al-Asbahi, my close friend.
- Mohammed Al-Merhabi, a dear friend to me.
- Every member in MSOMS community!

OhjyA is a unique algorithm!
Ghasan Al-Sakkaf